Crate commonware_runtime

Settings

Help

Summary

Source

Execute asynchronous tasks with a configurable scheduler.

This crate provides a collection of runtimes that can be used to execute asynchronous tasks in a variety of ways. For production use, the `tokio` module provides a runtime backed by Tokio. For testing and simulation, the `deterministic` module provides a runtime that allows for deterministic execution of tasks (given a fixed seed).

Status

`commonware-runtime` is ALPHA software and is not yet recommended for production use. Developers should expect breaking changes and occasional instability.

Modules

- deterministic
- A deterministic runtime that randomly selects tasks to run based on a seed
- mocks
- A mock implementation of a channel that implements the Sink and Stream traits.
- tokio
- A production-focused runtime based on Tokio with secure randomness and storage backed by the local filesystem.

Structs

- Handle
- Handle to a spawned task.
- Signaler
- Coordinates a one-time signal across many tasks.

Enums

- Error

Traits

- Blob
- Interface to read and write to a blob.
- Clock
- Interface that any task scheduler must implement to provide time-based operations.
- Listener
- Interface that any runtime must implement to handle incoming network connections.
- Network
- Interface that any runtime must implement to create network connections.
- Runner
- Interface that any task scheduler must implement to start running tasks.
- Sink
- Interface that any runtime must implement to send messages over a network connection.
- Spawner
- Interface that any task scheduler must implement to spawn sub-tasks in a given root task.
- Storage
- Interface to interact with storage.
- Stream
- Interface that any runtime must implement to receive messages over a network connection.

Functions

- reschedule
- Yield control back to the runtime.

Type Aliases

- Signal
- A one-time broadcast that can be awaited by many tasks. It is often used for coordinating shutdown across many tasks.

commonware_runtime

Module deterministic copy item path

Settings

Help

Summary

Source

A deterministic runtime that randomly selects tasks to run based on a seed

Panics

If any task panics, the runtime will panic (and shutdown).

Example

```
use commonware_runtime::{Spawner, Runner,
deterministic::Executor};

let (executor, runtime, auditor) = Executor::default();
executor.start(async move {
    println!("Parent started");
    let result = runtime.spawn("child", async move {
        println!("Child started");
        "hello"
    });
    println!("Child result: {:?}", result.await);
    println!("Parent exited");
});
  println!("Auditor state: {}", auditor.state());
```

Structs

- Auditor
- Track the state of the runtime for determinism auditing.
- Blob
- Implementation of `crate::Blob` for the `deterministic` runtime.
- Config
- Configuration for the `deterministic` runtime.
- Context

- Implementation of `crate::Spawner`, `crate::Clock`, `crate::Network`, and `crate::Storage` for the `deterministic` runtime.
- Executor
- Deterministic runtime that randomly selects tasks to run based on a seed.
- Listener
- Implementation of `crate::Listener` for the `deterministic` runtime.
- Runner
- Implementation of `crate::Runner` for the `deterministic` runtime.
- Sink
- Implementation of `crate::Sink` for the `deterministic` runtime.
- Stream
- Implementation of `crate::Stream` for the `deterministic` runtime.

Type Aliases

- Partition
- Map of names to blob contents.

commonware_runtime

Module mocks

Settings

Help

Summary

Source

A mock implementation of a channel that implements the Sink and Stream traits.

Structs

- Channel
- A mock channel struct that is used internally by Sink and Stream.
- Sink
- A mock sink that implements the Sink trait.
- Stream
- A mock stream that implements the Stream trait.

commonware_runtime

Module tokio

Settings

Help

Summary

Source

A production-focused runtime based on Tokio with secure randomness and storage backed by the local filesystem.

Panics

By default, the runtime will catch any panic and log the error. It is possible to override this behavior in the configuration.

Example

```
use commonware_runtime::{Spawner, Runner, tokio::Executor};

let (executor, runtime) = Executor::default();
executor.start(async move {
    println!("Parent started");
    let result = runtime.spawn("child", async move {
        println!("Child started");
        "hello"
    });
    println!("Child result: {:?}", result.await);
    println!("Parent exited");
});
```

Structs

- Blob
- Implementation of `crate::Blob` for the `tokio` runtime.
- Config
- Configuration for the `tokio` runtime.
- Context

- Implementation of `crate::Spawner`, `crate::Clock`, `crate::Network`, and `crate::Storage` for the `tokio` runtime.
- Executor
- Runtime based on Tokio.
- Listener
- Implementation of `crate::Listener` for the `tokio` runtime.
- Runner
- Implementation of `crate::Runner` for the `tokio` runtime.
- Sink
- Implementation of `crate::Sink` for the `tokio` runtime.
- Stream
- Implementation of `crate::Stream` for the `tokio` runtime.

commonware_runtime

Struct Handle

Copy item path

Settings

Help

Summary

Source

```
pub struct Handle<T>
where
    T: Send + 'static,
{ /* private fields */ }
```

Handle to a spawned task.

Implementations

Source

```
impl<T> Handle<T>

where
    T: Send + 'static,
```

Source

```
pub fn abort(&self)
```

Trait Implementations

Source

```
impl<T> Future for Handle<T>

where
    T: Send + 'static,
```

Source

```
type Output = Result<T, Error>
```

The type of value produced on completion.

Source

```
fn poll(self: Pin<&mut Self>, cx: &mut Context<'_>) ->
Poll<Self::Output>
```

Attempts to resolve the future to a final value, registering the current task for wakeup if the value is not yet available. Read more

Auto Trait Implementations

```
impl<T> Freeze for Handle<T>

impl<T> !RefUnwindSafe for Handle<T>

impl<T> Send for Handle<T>

impl<T> Sync for Handle<T>

impl<T> Unpin for Handle<T>

impl<T> !UnwindSafe for Handle<T>
```

Blanket Implementations

Source

```
impl<T> Any for T

where
```

```
    T: 'static + ?Sized,
```

Source

```
impl<T> Borrow<T> for T

where
    T: ?Sized,
```

Source

```
impl<T> BorrowMut<T> for T

where
    T: ?Sized,
```

Source

```
impl<T> From<T> for T
```

Source

```
impl<T> FutureExt for T

where
    T: Future + ?Sized,
```

Source

```
impl<T> Instrument for T
```

Source

```
impl<T, U> Into<U> for T

where
    U: From<T>,
```

Source

```
impl<F> IntoFuture for F

where
```

```
    F: Future,
```

Source

```
impl<T> Same for T
```

Source

§

```
impl<T, U> TryFrom<U> for T

where
    U: Into<T>,
```

Source

```
impl<F, T, E> TryFuture for F

where
    F: Future<Output = Result<T, E>> + ?Sized,
```

Source

```
impl<Fut> TryFutureExt for Fut

where
    Fut: TryFuture + ?Sized,
```

Source

```
impl<T, U> TryInto<U> for T

where
    U: TryFrom<T>,
```

Source

```
impl<V, T> VZip<V> for T

where
    V: MultiLane<T>,
```

Source

impl<T> WithSubscriber for T

commonware_runtime

Struct Signaler

Settings

Help

Summary

Source

```
pub struct Signaler { /* private fields */ }
```

Coordinates a one-time signal across many tasks.

Example

Basic Usage

```
use commonware_runtime::{Spawner, Runner, Signaler,
deterministic::Executor};

let (executor, _, _) = Executor::default();
executor.start(async move {
    // Setup signaler and get future
    let (mut signaler, signal) = Signaler::new();

    // Signal shutdown
    signaler.signal(2);

    // Wait for shutdown in task
    let sig = signal.await.unwrap();
    println!("Received signal: {}", sig);
});
```

Advanced Usage

While `Futures::Shared` is efficient, there is still meaningful overhead to cloning it (i.e. in each iteration of a loop). To avoid a performance regression from introducing `Signaler`, it is recommended to wait on a reference to `Signal` (i.e. `&mut signal`).

```
use commonware_macros::select;
```

```
use commonware_runtime::{Clock, Spawner, Runner, Signaler,
deterministic::Executor};
use futures::channel::oneshot;
use std::time::Duration;

let (executor, context, _) = Executor::default();
executor.start(async move {
    // Setup signaler and get future
    let (mut signaler, mut signal) = Signaler::new();

    // Loop on the signal until resolved
    let (tx, rx) = oneshot::channel();
    context.spawn("task", {
        let context = context.clone();
        async move {
            loop {
                // Wait for signal or sleep
                select! {
                     sig = &mut signal => {
                         println!("Received signal: {}",
sig.unwrap());
                         break;
                     },
                     _ = context.sleep(Duration::from_secs(1)) =>
{},
                };
            }
            let _ = tx.send(());
        }
    });

    // Send signal
    signaler.signal(9);

    // Wait for task
    rx.await.expect("shutdown signaled");
  });
```

Implementations

Source

impl Signaler

Source

```
pub fn new() -> (Self, Signal)
```

Create a new `Signaler`.

Returns a `Signaler` and a `Signal` that will resolve when `signal` is called.

Source

```
pub fn signal(&mut self, value: i32)
```

Resolve the `Signal` for all waiters (if not already resolved).

Auto Trait Implementations

```
impl Freeze for Signaler
impl !RefUnwindSafe for Signaler
impl Send for Signaler
impl Sync for Signaler
impl Unpin for Signaler
impl !UnwindSafe for Signaler
```

Blanket Implementations

Source

```
impl<T> Any for T

where
    T: 'static + ?Sized,
```

Source

```
impl<T> Borrow<T> for T

where
    T: ?Sized,
```

Source

```
impl<T> BorrowMut<T> for T

where
    T: ?Sized,
```

Source

```
impl<T> From<T> for T
```

Source

```
impl<T> Instrument for T
```

Source

```
impl<T, U> Into<U> for T

where
    U: From<T>,
```

Source

```
impl<T> Same for T
```

Source

```
impl<T, U> TryFrom<U> for T

where
    U: Into<T>,
```

Source

```
impl<T, U> TryInto<U> for T

where
    U: TryFrom<T>,
```

Source

```
impl<V, T> VZip<V> for T

where
```

```
    V: MultiLane<T>,
```

Source

```
impl<T> WithSubscriber for T
```

commonware_runtime

Enum Error Copy item path

Settings

Help

Summary

Source

```
pub enum Error {
```
Show 19 variants}

Variants

Exited

Closed

Timeout

BindFailed

ConnectionFailed

WriteFailed

ReadFailed

SendFailed

RecvFailed

PartitionCreationFailed(String)

PartitionMissing(String)

PartitionCorrupt(String)

BlobOpenFailed(String, String)

BlobMissing(String, String)

BlobTruncateFailed(String, String)

BlobSyncFailed(String, String)

BlobCloseFailed(String, String)

BlobInsufficientLength

OffsetOverflow

Trait Implementations

Source

impl Debug for Error

Source

fn fmt(&self, f: &mut Formatter<'_>) -> Result

Formats the value using the given formatter. Read more

Source

impl Display for Error

Source

fn fmt(&self, __formatter: &mut Formatter<'_>) -> Result

Formats the value using the given formatter. Read more

Source

`impl Error for Error`

1.30.0 · Source

```
fn source(&self) -> Option<&(dyn Error + 'static)>
```

Returns the lower-level source of this error, if any. Read more

1.0.0 · Source

```
fn description(&self) -> &str
```

👎Deprecated since 1.42.0: use the Display impl or to_string()

Read more

1.0.0 · Source

```
fn cause(&self) -> Option<&dyn Error>
```

👎Deprecated since 1.33.0: replaced by Error::source, which can support downcasting

Source

```
fn provide<'a>(&'a self, request: &mut Request<'a>)
```

🔬This is a nightly-only experimental API. (`error_generic_member_access`)

Provides type-based access to context intended for error reports. Read more

Source

```
impl PartialEq for Error
```

Source

```
fn eq(&self, other: &Error) -> bool
```

Tests for `self` and `other` values to be equal, and is used by `==`.

1.0.0 · Source

```
fn ne(&self, other: &Rhs) -> bool
```

Tests for `!=`. The default implementation is almost always sufficient, and should not be overridden without very good reason.

Source

```
impl StructuralPartialEq for Error
```

Auto Trait Implementations

```
impl Freeze for Error
impl RefUnwindSafe for Error
impl Send for Error
impl Sync for Error
impl Unpin for Error
impl UnwindSafe for Error
```

Blanket Implementations

Source

```
impl<T> Any for T

where
    T: 'static + ?Sized,
```

Source

```
impl<T> Borrow<T> for T

where
    T: ?Sized,
```

Source

```
impl<T> BorrowMut<T> for T

where
    T: ?Sized,
```

Source

```
impl<T> From<T> for T
```

Source

```
impl<T> Instrument for T
```

Source

```
impl<T, U> Into<U> for T

where
    U: From<T>,
```

Source

```
impl<T> Same for T
```

Source

```
impl<T> ToString for T

where
    T: Display + ?Sized,
```

Source

```
impl<T, U> TryFrom<U> for T

where
    U: Into<T>,
```

Source

```
impl<T, U> TryInto<U> for T

where
    U: TryFrom<T>,
```

Source

```
impl<V, T> VZip<V> for T

where
    V: MultiLane<T>,
```

Source

```
impl<T> WithSubscriber for T
```

commonware_runtime

Trait Blob Copy item path

Settings

Help

Summary

Source

```
pub trait Blob:
    Clone
    + Send
    + Sync
    + 'static {
    // Required methods
    fn len(&self) -> impl Future<Output = Result<u64, Error>>
+ Send;

   fn read_at(
        &self,
        buf: &mut [u8],
        offset: u64,
    ) -> impl Future<Output = Result<(), Error>> + Send;

   fn write_at(
        &self,
        buf: &[u8],
        offset: u64,
    ) -> impl Future<Output = Result<(), Error>> + Send;

   fn truncate(
        &self,
        len: u64,
    ) -> impl Future<Output = Result<(), Error>> + Send;

   fn sync(&self) -> impl Future<Output = Result<(), Error>> +
Send;

   fn close(self) -> impl Future<Output = Result<(), Error>> +
Send;
}
```

Interface to read and write to a blob.

To support blob implementations that enable concurrent reads and writes, blobs are responsible for maintaining synchronization.

Cloning a blob is similar to wrapping a single file descriptor in a lock whereas opening a new blob (of the same name) is similar to opening a new file descriptor. If multiple blobs are opened with the same name, they are not expected to coordinate access to underlying storage and writing to both is undefined behavior.

Required Methods

Source

```
fn len(&self) -> impl Future<Output = Result<u64, Error>> +
Send
```

Get the length of the blob.

Source

```
fn read_at(
    &self,
    buf: &mut [u8],
    offset: u64,
) -> impl Future<Output = Result<(), Error>> + Send
```

Read from the blob at the given offset.

`read_at` does not return the number of bytes read because it only returns once the entire buffer has been filled.

Source

```
fn write_at(
    &self,
    buf: &[u8],
    offset: u64,
) -> impl Future<Output = Result<(), Error>> + Send
```

Write to the blob at the given offset.

Source

```
fn truncate(&self, len: u64) -> impl Future<Output =
Result<(), Error>> + Send
```

Truncate the blob to the given length.

Source

```
fn sync(&self) -> impl Future<Output = Result<(), Error>> +
Send
```

Ensure all pending data is durably persisted.

Source

```
fn close(self) -> impl Future<Output = Result<(), Error>> +
Send
```

Close the blob.

Dyn Compatibility

§

This trait is not dyn compatible.

In older versions of Rust, dyn compatibility was called "object safety", so this trait is not object safe.

Implementors

Source

impl Blob for commonware_runtime::deterministic::Blob

Source

impl Blob for commonware_runtime::tokio::Blob

commonware_runtime

Trait Clock

Settings

Help

Summary

Source

```
pub trait Clock:
    Clone
    + Send
    + Sync
    + 'static {
    // Required methods
    fn current(&self) -> SystemTime;

   fn sleep(
        &self,
        duration: Duration,
    ) -> impl Future<Output = ()> + Send + 'static;

   fn sleep_until(
        &self,
        deadline: SystemTime,
    ) -> impl Future<Output = ()> + Send + 'static;
}
```

Interface that any task scheduler must implement to provide time-based operations.

It is necessary to mock time to provide deterministic execution of arbitrary tasks.

Required Methods

Source

`fn current(&self) -> SystemTime`

Returns the current time.

Source

```
fn sleep(&self, duration: Duration) -> impl Future<Output =
()> + Send + 'static
```

Sleep for the given duration.

Source

```
fn sleep_until(
    &self,
    deadline: SystemTime,
) -> impl Future<Output = ()> + Send + 'static
```

Sleep until the given deadline.

Dyn Compatibility

This trait is not dyn compatible.

In older versions of Rust, dyn compatibility was called "object safety", so this trait is not object safe.

Implementors

Source

impl Clock for commonware_runtime::deterministic::Context

Source

impl Clock for commonware_runtime::tokio::Context

commonware_runtime

Trait Listener

Copy item path

Settings

Help

Summary

Source

```
pub trait Listener<Si, St>:
    Sync
    + Send
    + 'static
where
    Si: Sink,
    St: Stream,
{
    // Required method
    fn accept(
        &mut self,
    ) -> impl Future<Output = Result<(SocketAddr, Si, St),
Error>> + Send;
}
```

Interface that any runtime must implement to handle incoming network connections.

Required Methods

Source

```
fn accept(
    &mut self,
) -> impl Future<Output = Result<(SocketAddr, Si, St), Error>>
+ Send
```

Accept an incoming connection.

Dyn Compatibility

This trait is not dyn compatible.

In older versions of Rust, dyn compatibility was called "object safety", so this trait is not object safe.

Implementors

Source

impl Listener<Sink, Stream> for commonware_runtime::deterministic::Listener

Source

impl Listener<Sink, Stream> for commonware_runtime::tokio::Listener

commonware_runtime

Trait Network

Settings

Help

Summary

Source

```
pub trait Network<L, Si, St>:
    Clone
    + Send
    + Sync
    + 'static
where
    L: Listener<Si, St>,
    Si: Sink,
    St: Stream,
{
    // Required methods
    fn bind(
        &self,
        socket: SocketAddr,
    ) -> impl Future<Output = Result<L, Error>> + Send;

   fn dial(
        &self,
        socket: SocketAddr,
    ) -> impl Future<Output = Result<(Si, St), Error>> + Send;
}
```

Interface that any runtime must implement to create network connections.

Required Methods

Source

```
fn bind(
    &self,
    socket: SocketAddr,
) -> impl Future<Output = Result<L, Error>> + Send
```

Bind to the given socket address.

Source

```
fn dial(
    &self,
    socket: SocketAddr,
) -> impl Future<Output = Result<(Si, St), Error>> + Send
```

Dial the given socket address.

Dyn Compatibility

This trait is not dyn compatible.

In older versions of Rust, dyn compatibility was called "object safety", so this trait is not object safe.

Implementors

§

Source

impl Network<Listener, Sink, Stream> for commonware_runtime::deterministic::Context

Source

impl Network<Listener, Sink, Stream> for commonware_runtime::tokio::Context

commonware_runtime

Trait Runner

Settings

Help

Summary

Source

```
pub trait Runner {
    // Required method
    fn start<F>(self, f: F) -> F::Output
       where F: Future + Send + 'static,
             F::Output: Send + 'static;
}
```

Interface that any task scheduler must implement to start running tasks.

Required Methods

Source

```
fn start<F>(self, f: F) -> F::Output
where
    F: Future + Send + 'static,
    F::Output: Send + 'static,
```

Start running a root task.

Dyn Compatibility

This trait is not dyn compatible.

In older versions of Rust, dyn compatibility was called "object safety", so this trait is not object safe.

Implementors

Source

impl Runner for commonware_runtime::deterministic::Runner

Source

impl Runner for commonware_runtime::tokio::Runner

commonware_runtime

Trait Sink

Settings

Help

Summary

Source

```
pub trait Sink:
    Sync
    + Send
    + 'static {
    // Required method
    fn send(
        &mut self,
        msg: &[u8],
    ) -> impl Future<Output = Result<(), Error>> + Send;
}
```

Interface that any runtime must implement to send messages over a network connection.

Required Methods

Source

fn send(&mut self, msg: &[u8]) -> impl Future<Output = Result<(), Error>> + Send

Send a message to the sink.

Dyn Compatibility

This trait is not dyn compatible.

In older versions of Rust, dyn compatibility was called "object safety", so this trait is not object safe.

Implementors

Source

impl Sink for commonware_runtime::deterministic::Sink

Source

impl Sink for commonware_runtime::mocks::Sink

Source

impl Sink for commonware_runtime::tokio::Sink

commonware_runtime

Trait Spawner

Settings

Help

Summary

Source

```
pub trait Spawner:
    Clone
    + Send
    + Sync
    + 'static {
    // Required methods
    fn spawn<F, T>(&self, label: &str, f: F) -> Handle<T> ⓘ
       where F: Future<Output = T> + Send + 'static,
             T: Send + 'static;

   fn stop(&self, value: i32);

   fn stopped(&self) -> Signal;
}
```

Interface that any task scheduler must implement to spawn sub-tasks in a given root task.

Required Methods

Source

```
fn spawn<F, T>(&self, label: &str, f: F) -> Handle<T> ⓘ
where
    F: Future<Output = T> + Send + 'static,
    T: Send + 'static,
```

Enqueues a task to be executed.

Label can be used to track how many instances of a specific type of task have been spawned or are running concurrently (and is appened to all metrics). Label is automatially appended to the parent task labels (i.e. spawning “fun” from “have” will be labeled “have_fun”).

Unlike a future, a spawned task will start executing immediately (even if the caller does not await the handle).

Source

```
fn stop(&self, value: i32)
```

Signals the runtime to stop execution and that all outstanding tasks should perform any required cleanup and exit. This method is idempotent and can be called multiple times.

This method does not actually kill any tasks but rather signals to them, using the `Signal` returned by `stopped`, that they should exit.

Source

```
fn stopped(&self) -> Signal
```

Returns an instance of a `Signal` that resolves when `stop` is called by any task.

If `stop` has already been called, the returned `Signal` will resolve immediately.

Dyn Compatibility

This trait is not dyn compatible.

In older versions of Rust, dyn compatibility was called "object safety", so this trait is not object safe.

Implementors

Source

impl Spawner for commonware_runtime::deterministic::Context

Source

impl Spawner for commonware_runtime::tokio::Context

commonware_runtime

Trait Storage

Settings

Help

Summary

Source

```
pub trait Storage<B>:
    Clone
    + Send
    + Sync
    + 'static
where
    B: Blob,
{
    // Required methods
    fn open(
```

```
        &self,
        partition: &str,
        name: &[u8],
    ) -> impl Future<Output = Result<B, Error>> + Send;

    fn remove(
        &self,
        partition: &str,
        name: Option<&[u8]>,
    ) -> impl Future<Output = Result<(), Error>> + Send;

    fn scan(
        &self,
        partition: &str,
    ) -> impl Future<Output = Result<Vec<Vec<u8>>, Error>> +
Send;
}
```

Interface to interact with storage.

To support storage implementations that enable concurrent reads and writes, blobs are responsible for maintaining synchronization.

Storage can be backed by a local filesystem, cloud storage, etc.

Required Methods

Source

```
fn open(
    &self,
    partition: &str,
    name: &[u8],
) -> impl Future<Output = Result<B, Error>> + Send
```

Open an existing blob in a given partition or create a new one.

Multiple instances of the same blob can be opened concurrently, however, writing to the same blob concurrently may lead to undefined behavior.

Source

```
fn remove(
    &self,
    partition: &str,
    name: Option<&[u8]>,
) -> impl Future<Output = Result<(), Error>> + Send
```

Remove a blob from a given partition.

If no `name` is provided, the entire partition is removed.

Source

```
fn scan(
    &self,
    partition: &str,
) -> impl Future<Output = Result<Vec<Vec<u8>>, Error>> + Send
```

Return all blobs in a given partition.

Dyn Compatibility

This trait is not dyn compatible.

In older versions of Rust, dyn compatibility was called "object safety", so this trait is not object safe.

Implementors

Source

impl Storage<Blob> for commonware_runtime::deterministic::Context

Source

impl Storage<Blob> for commonware_runtime::tokio::Context

commonware_runtime

Trait Stream

Settings

Help

Summary

Source

```
pub trait Stream:
    Sync
    + Send
    + 'static {
    // Required method
    fn recv(
        &mut self,
        buf: &mut [u8],
    ) -> impl Future<Output = Result<(), Error>> + Send;
}
```

Interface that any runtime must implement to receive messages over a network connection.

Required Methods

Source

```
fn recv(
    &mut self,
    buf: &mut [u8],
) -> impl Future<Output = Result<(), Error>> + Send
```

Receive a message from the stream, storing it in the given buffer. Reads exactly the number of bytes that fit in the buffer.

Dyn Compatibility

This trait is not dyn compatible.

In older versions of Rust, dyn compatibility was called "object safety", so this trait is not object safe.

Implementors

Source

impl Stream for commonware_runtime::deterministic::Stream

Source

impl Stream for commonware_runtime::mocks::Stream

Source

impl Stream for commonware_runtime::tokio::Stream